UNIVERSITY OF AMSTERDAM

A Fault Tolerance Framework in a Concurrent Programming Environment

J. Fu

Summary

As CMOS technology scales ever further, multi-core processors are becoming mainstream both in research and industry. However, the system vulnerability is increasing due to tighter design margins and greater susceptibility to interference, both caused by smaller feature size, lower power supply voltage, higher frequency, greater hardware complexity and more transistors per processor. Meanwhile, concurrent programming environment has emerged, a general designation for the norms in the exploitation of systems with multi-core processors, which is widely believed to be the main approach for gaining scalable performance improvement from multi-core systems based on parallelism exploitation and resource scheduling. In this dissertation, we specifically explore the construction of a fault tolerance framework in a concurrent programming environment. During this process, we investigate the features of a concurrent programming environment. With this knowledge, we design a cross-layer, flexible, low-overhead fault tolerance framework including fault detection, and recovery, as well as fault injection for its evaluation. The proposed fault tolerance framework targets the general paradigm of concurrent programming environments, and is evaluated and implemented in a specific platform, i.e., the Microgrids.

This dissertation includes three main parts: fault detection, recovery and injection in a concurrent programming environment. In the fault detection part, we present on-demand, thread-level redundancy in the context of a concurrent programming environment. When and where a program should be duplicated to give high reliability can be specified by programmers or the run-time environment. This makes the system more efficient and flexible through providing affordable fault tolerance based on the system's current situation. Meanwhile, we also evaluate the redundant multithreading (RMT) technique in a data-flow scheduled multithreaded (DMT) environment, which is motivated by the evaluation of on-demand redundancy and the DMT implementation in the Microgrids. The results confirm that RMT can benefit from DMT. Furthermore, we provide an asynchronous output comparison mechanism for core-level redundant execution technique, which uses a shared buffer for output comparison between a fixed-pair core. It avoids the inter-core data transmission for comparison and saves the capacity of buffer.

In the fault recovery part, we uncover an opportunity to reduce the overheads of fault recovery dramatically in modern processors that appears as a side-effect of introducing hardware multithreading to improve performance. This is because, in our assumed model, threads are usually short code sequences with no branches and few memory side-effects, which means that the number of checkpoints is small and constant. In addition, the state structures of a thread already presented in

hardware can be reused to provide checkpointing. We demonstrate this principle of using a hardware/software co-design called *Rethread*, which features compiler-generated code annotations and automatic recovery in hardware by re-executing threads. *Rethread* is a quite novel strategy with extremely low hardware and performance overheads, since it makes full use of the features of multithreading environment and fault detection mechanism.

In the fault injection part, we propose a micro-architecture level simulator implemented fault injection technique for dependability analysis. Based on this fault injection technique, we analyzes the vulnerability of different microarchitectural structures in the Microgrids, especially for the structures used to support hardware concurrency management. Also, we quantify the level of dependability that the fault tolerance approaches proposed in previous parts provide. Finally, we also provide the correlation between the micro-architecture level faults and their impact on the application level. The dependability analysis is not only a valuable reference for designing a reliable concurrent programming environment, but also an important argument for whether implementing concurrency management in hardware or software.

Overall, the fault tolerance framework proposed in this dissertation is an enhancement of the concurrent programming environments. There are two related work directions: one is augmented with fault injection; another is enriched with fault detection and fault recovery progressively. Finally, the fault injection enhancement is used to analyze the dependability of the concurrent programming environments, and measure the capability of the proposed fault detection and recovery strategies. Furthermore, the fault tolerance framework is quite comprehensive and universal, which includes fault detection, fault recovery and fault injection. It is also very flexible and efficient by making full use of some features of the concurrent programming environments, such as the thread-level concurrency expression.